Niche Search: an Evolutionary Algorithm for Global Optimisation

João Pedro Pedroso

Centre for Operations Research and Econometrics
34 Voie du Roman Pays,
B-1348 Louvain-la-Neuve, Belgium

Abstract. In this paper we describe niche search, a genetic-based optimisation approach which is characterised by an evolutionary search on two layers: the individual layer (which is comparable to search described in other genetic algorithms), and the niche layer.
Neither of these searches is directed: both individuals and niches evolve based on the selection of the fittest.
The numerical results obtained by niche search are quite promising, as our implementation has successfully handled all the tests carried out. The computational performance is considerably better than that of other algorithms of the same family analysed in the literature.

1 Overview

Niche search is a genetic algorithm version for whose conception special attention has been paid to the aspects that follow.

First, encode the reproduction operations that occur in nature, at the cellular and molecular level, more precisely and completely than the implementation proposed in the canonical genetic algorithm (CGA). In particular, it is proposed a simulation of the *meiosis* process in the reproduction of our artificial organisms. Meiosis is the main source of diversity in nature (for sexual reproduction), and is absent in the CGA.

Second, the overall population of individuals — the *ecosystem* — is assembled into different groups, each of them occupying a particular *niche*. This parallels what happens in nature: individuals of a given species are in general not found neither alone nor assembled in the same global pool. Furthermore, the parameters that control the reproduction (mutation probability, crossover probability, etc.) may vary from place to place, as may vary the way in which different species exploit different parameters. In the approach proposed, each niche is supplied its own set of these parameters by the ecosystem, in an arbitrary and endogenous way.

Finally, selection and reproduction act not only at the level of individuals, but also at the level of groups of individuals (niches). Individuals must compete in order to survive among their similars; niches must also compete between them for resources in the environment. Competition and selection between individuals of a particular niche is similar to what is proposed in the CGA; competition

between niches is, to the best of our knowledge, proposed here the first time. *Niche selection* means that, at each generation, there are some poorly adapted niches which may disappear; niches whose adaptation is good remain.

Hence, niche search captures issues concerning evolution on both individual organisms and groups, or organisations of individuals. A diagram showing the general structure of the algorithm is presented in figure 1. The innovation, from the point of view of numerical optimisation, is that the adaptation of the main control parameters of the algorithm is made through an evolutionary approach — in contrast to what happens in evolutionary systems (ESs) and in evolutionary programming (EP), where auto adaption is directed by the program, based on the status of the system at each iteration. Our approach is also different from the one proposed by the *multilevel genetic algorithm* [8, 4].

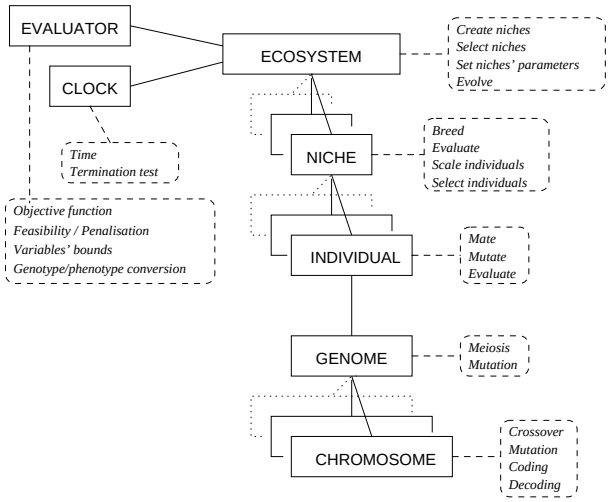

Fig. 1. Overview of the data structures and operators for niche search.

2 Niche search

2.1 The ecosystem and niches' evolution

What characterises a niche in an ecosystem is its set of individuals and the parameters of the environment where the niche is located (the reproduction factors). Parameters are imposed to the niches externally (and randomly) by the ecosystem; nothing directs niches to find the "best" reproduction parameters. Niches for which the parameters are good are likely to adapt faster to the pressures of the environment, and hence remain alive; those whose parameters are bad, are likely to adapt slower, and possibly extinguish.

Reproduction parameters imposed to the niches are: probability and intensity of mutation of the genes; probability and intensity of occurrence of crossover in the chromosome replication, and a selectivity factor (a factor that specifies how competitive an individual must be in relation to the average in order to have a favoured reproduction probability). Notice that the CGA corresponds to a single niche, where these parameters are set exogenously. The performance of that algorithm for difficult problems deeply depends on them; in niche search, they are endogenous, resulting from of an "evolutionary search".

Evolution at the niches level is based on the following operators (which parallel the operators found at the individual level):

Niche selection. Poorly adapted niches are likely to be extinguished, whilst better adapted ones remain.

Niche recombination. Niches which disappear are replaced by new ones; these emerge as a selection of individuals that "migrated" from existing niches.

Niche mutation. The parameters that characterise each niche are randomly changed (mutated) at each generation. This operator is inspired by the fact that the equivalent parameters continuously change the in natural life.

Pseudo programming code explaining the global operation of the niche search algorithm is presented in figure 2.

2.2 Niches and individual's evolution

Inside each niche, individuals are submitted to competition between their similars — i.e., with the other individuals of the niche. Selection and reproduction inside a niche follow the scheme proposed in the CGA, except in the following factors:

1. If a new niche, created with the reproduction parameters that the ecosystem supplies, is better than the progenitor niche, it can breed again (up to a maximum number of times) in the same "generation". The biological interpretation for this, is that when natural conditions are better, organisms tend to reproduce more and faster.
2. Not every new population generated is accepted. If both the average and the best individual are worse that their counterparts in the progenitor niche, then, with some probability, the offspring niche is rejected, and the old niche remains unchanged for the next generation[1].
3. We allow for the possibility of representing the chromosome of an individual by data structures other than bit strings. Our implementation supports both bit string and floating point genotypes.

[1] The natural interpretation for this in intuitive. Notice that this equivalent to what happens in Simulated Annealing: there, new solutions are accepted whenever there is an improvement, and may are rejected, with a given probability, when they are worse.

```
set t := 0                                          Start with an initial time.
set ecosystem(t) := InitEcosystem()                 Initialise the ecosystem:
    niches(t) = CreateNiches(t)                     Create the desired number of niches for the run
    InitParameters(niches(t))                       Randomly initialise the parameters that characterise each niche:
                                                    crossover probability, mutation probability, etc.
    InitialisePopulation(niches(t))                 Randomly initialise the population of each niche.
Evaluate(niches(t))                                 Evaluate the fitness of all the niches in the initial population. We propose an
                                                    evaluation based on two parameters: the average fitness of the niches' population and the fitness of its
                                                    best individual, and we use the sum of these factors as the niche's overall evaluation.
iterate                                             Start the evolution process.
    strong(t) := SelectStrong(ecosystem(t))         Select the niches that will survive to the next
                                                    generation.
    weak(t) := SelectWeak(ecosystem(t))             Select the niches that may extinguish.
    newniches(t) := Recombination(weak(t),strong(t))  With some probability, replace the
                                                    populations of weak niches by a selection of individuals from the union of that niche with a strong
                                                    one.
    InitParameters(newniches(t))                    Randomly initialise the parameters of the created niches.
    Evaluate(newniches(t))                          Evaluate the new niches.
    niches(t+1) := Selection(niches(t), new(t))     The new ecosystem is a selection of the best
                                                    niches. In our implementation, we simply replace the the weak niches by the new ones.
    MutateParameters(niches(t+1))                   With some probability, randomly perturb the parameters of
                                                    each niche.
    Breed(niches(t+1))                              Create a new generation of individuals in each of the niches.
    t := t + 1                                      Increase the time counter.
until Terminated()                                  Termination criteria: time.
display solution                                    Solution is the best individual found.
```

Fig. 2. Niche Search: pseudo code for the outer level

4. Niche search provides two parameters for mutation and two parameters for crossover, for controlling mating between individuals. One specifies the probability of occurrence of these operators, and the other determines its intensity.

The possibility of representing genotypes different from bit strings was suggested by several authors. In [5], the bit string representation receives considerable criticism: the precision of solutions represented by bit strings is considerably limited for a low number of bits, and the convergence for a large number of bits is rather slow. The success of the direct floating point representation of the phenotypes in ESs and EP also suggests the possibility of representing the chromosome as a floating point; in our implementation, floating point genotypes are values in the interval $[0, 1]$, whose phenotype can be assessed by scaling these values to the region defined by the lower and upper bound of the corresponding variable.

Mutation of individuals is based on two parameters: the mutation probability (which specifies the probability of a given chromosome being mutated) and the mutation intensity. This last, controls the magnitude of the mutation. In bit strings, this corresponds to the likelihood of any of the genes of the chromosome suffering a mutation. In floating point genomes this corresponds to a measure of

the deviation that a mutation may produce, in relation to the original value.

On the same way, two parameters control the crossover: the crossover probability and the crossover intensity. On bit strings, higher intensity increases the number of potential mutation points (and hence the amount of "shuffling" between the two chromosomes). On floating point strings, crossover intensity specifies the likelihood of obtaining a solution which is intermediary between the two original values, instead of the value of either of them unchanged.

For pseudo programming code explaining the evolution process inside a niche, please refer to figure 3.

```
g := 0                                          Initialise the "subgeneration" counter.
subpopulation(g) := population(t)               Select subpopulation for offspring production, i.e. the subset of
                                                the niche's elements that will be able to mate. (t is the global generation counter, in the ecosystem)
iterate                                         Start the evolution process.
    parents(g) := Selection(subpopulation(g))   Select parents for offspring production, e.g.
                                                through roulette wheel selection.
    offspring(g) := Recombination(parents(g))   Recombine the genomes of two selected
                                                parents, through the meiosis operator (figure 4).
    offspring(g) := Mutation(offspring(g))      With some probability, randomly perturb each of the
                                                chromosomes of each individual.
    Evaluate(offspring(g))                      Evaluate the objective of all the individuals in the niche's population.
                                                Penalise and scale; obtain the fitnesses.
    subpopulation(g+1) := offspring(g)          New subpopulation: the offspring of the previous one.
    g := g + 1                                  Increase the subgeneration counter.
    population(t+1) := ChooseOne(population(t),subpopulation(g))   Update the
                                                niche's population. Always select the subpopulation is its performance is better; if the performance is
                                                worse, accept it with some probability.
until Terminated()                              Termination criteria: maximum subgenerations achieved, or evaluation of current
                                                subpopulation is worse than evaluation of last subgeneration's one.
```

Fig. 3. Niche search: pseudo code for the inner evolution (breeding) of a niche. This procedure is executed at each generation step of the ecosystem (the Breed() step, in figure 2).

2.3 Other differences between niche search and other GAs

Meiosis In niche search, the usual mutation and crossover operators on the genetic structure are complemented by a meiosis process, which is absent in other GAs. It performed as described in figure 4.

We view the meiosis operator as a considerable improvement over other GAs for the treatment of problems of larger dimension. The standard approach on GAs for multidimensional optimisation is to assemble all the components of a vector into a single chromosome string, loosing therefore the "independence" between the several chromosomes that our system provides, throught the association of each variable to a particular chromosome.

```
input(mother,father)      The originating (diploid) cell contains two copies of each of the chromosomes;
                                  one coming from the father and the other coming from the mother.
for c = 1 to NumerOfChromosomes() do                                  Start the meiosis process.
    origin := RandomChoose(FromMother(),FromFather())      Randomly select the
                                  chromosome; either from the father's or from the mother's copy.
    if origin ?= FromMother()
        source(c) := mother(c)                                  This chromosome comes from the mother.
        genome(c) := Crossover(source(c), father(c))   With some probability, crossover with
                                                                  the other chromosome occurs.
    else
        source(c) := father(c)                                  This chromosome comes from the father.
        genome(c) := Crossover(source(c), mother(c))   With some probability, crossover
                                                                  with the other chromosome occurs.
    end if
done                                                  Until all chromosomes in the genome are done.
return genome                                                 Haploid cell (gamete) is created.
```

Fig. 4. Niche search: the meiosis process.

Time adaption The aim of niche search is to produce the best and most accurate achievable solution in a limited amount of generations. In order to get more precise solutions at the evolution goes on, we propose a way of gradually decreasing the intensity of the perturbations that the search operators produce. This allows a refinement of the solution obtained, as the number of performed generations gets closer the total number of generations. We named this feature *time adaption*.

Time adaption is optional: for any particular run, the user can select if it is performed or not. If not, the mutation and crossover intensities and the selectivity level are the same from the beginning until the end of a program run. This is what generally happens in other implementations of GAs. If there is time adaption, the intensity parameters are adapted at each generation, as follows:

- mutation intensity decreases proportionally to the time elapsed, starting with the value of the corresponding parameter set by the ecosystem and being close to zero at the end of the evolutionary run;
- crossover intensity and selectivity level start with values close to zero and increase proportionally to the time elapsed, being close to the values set by the ecosystem at the end.

This adaptation of the parameters allows for an increasing perfection of the individuals, leading to offspring that are closer and closer to their progenitors; at the end, the diversity in each of the niches is rather small. We expect to be, then, close to achieving species perfectly adapted to the environment. From another point of view, we expect to be close to the global optimum of the objective function.

The time adaption factor may take on different functions of the elapsed

generations. Time adaption may be the most important factor for explaining the high precision of the solutions that niche search obtains.

Fitness: objective function and penalisation When the solution corresponding to a particular individual is outside the feasible region, the evaluation is composed of the objective function plus a penalty, which is supposed to be a value proportional to the severity of the infeasibility.

This value of the fitness for penalised individuals is the objective minus a value proportional to the penalty. It is computed as follows:

$$f'(x) = f(x) - (M - m) \cdot (1 + p(x) \cdot t) \tag{1}$$

where f' is the fitness, f is the objective function, M and m are respectively the maximum and minimum fitness for all the individuals in the population of the ecosystem at the current generation, $p(x)$ is the penalty, t is proportional to the number of generations elapsed.

Fitnesses determined this way guarantee that if there is any feasible solution, its fitness is higher than that of any infeasible one. As a consequence, if there is a feasible solution in the population, the "elite" will be feasible.

Computing the fitness this way allows infeasible individuals to survive at the beginning, while being progressively eliminated with the evolution process. This method proves to be quite interesting, specially when the optimum is found in a very narrow feasible region.

Generalised power law scaling Another difference between the niche search implementation and other GAs concerns the scaling operation.

The scaling method that we propose avoids the constraint, usual in other scaling procedures, that the objective functions must be positive in all the search domain. Another advantage is that it is independent of the absolute value of the fitness: adding a constant to all the fitnesses does not produce any change in the scaled value.

Scaled fitnesses are obtained as follow:

$$f''(x) = \left[\epsilon + \left(\frac{f'(x) - m}{M - m}\right) \cdot (1 - \epsilon)\right]^{\sigma} \tag{2}$$

where $f''(x)$ is the scaled fitness, $f'(x)$ is the (penalised) fitness, $\epsilon = 1/N$, m and M are minimum and maximum (also penalised) fitnesses in the niche, σ is the scaling power.

The first step consists on the normalisation of all the fitness values into the interval $[1/N, 1]$, where N is the number of elements in the niche. The second step consists on elevating this value to a power, greater or equal to zero, which is the *scaling power*. If there is time adaption, this value is itself the product of the selectivity parameter of the niche by the time adaption factor. Time adaption makes the niches more and more selective, as time goes on: at the beginning the selection is almost purely stochastic, whilst at the end a much increased weight is put on the best individuals.

The scaled fitness of the individual is used at the time of the selection.

Elitist niches In order for an evolutionary process to converge to the global optimum, it must assure that the best solution found always remains in the population. This is done through elitism.

In niche search, the ecosystem keeps the best solution found by all the niches. Further, we implement elitist modes in niches, which allow some of them to keep their best solution unchanged in their population.

By default, all the niches are *non-elitist* except two: the one which detains the best individual, and the one whose average is the best (which can be the previous one). The non-elitist niches can be performing their search in regions quite far away from the elitist ones; this proves to be a very powerful way of avoiding local optima. At the same time, the elitist niches perform a more localised search, in the neighbourhood of their best element.

3 Numerical tests

As our implementation provides the first results using the niche search approach, we found it necessary to proceed to an extensive set of tests. We relied on test functions that are commonly used by the evolutionary computation community, which are supposed to put in evidence the strengths and weaknesses of each implementation.

We have implemented the all test functions proposed by De Jong in his thesis dissertation [3], Schaffer's binary F6 function [7]. Each of the tests has been run at least 10 times. In none of the runs for each of the tests has the program failed to obtain the global optimum, in a considerably low number of function evaluations.

The sphere model, and Ackley's test function have also been implemented, the results being reported below. The solutions reported are the average of 10 runs performed, and the log of the evolution of the solution is that of an intermediate run. We have performed more than 100 runs for each of these tests; niche search has always found the global optimum.

3.1 Sphere model

This test consists of an extension of De Jong's first test function to 30 variables, with a wider search range:

$$\begin{gathered}\text{maximise } f(x) = -\sum_{i=1}^{n} x_i^2 \\ x_i \in [-30, 30], \quad n = 30\end{gathered}$$

Parameters used by niche search for this test: 750 generations, 3 niches with 3 individuals each. Average value of the solution found: $\bar{f}^* = -6.19 \cdot 10^{-04}$, $|\bar{x}_i^*| \leq 4.7 \cdot 10^{-3}$. Average number of function evaluations per run: 9089.

Experimental results for this test function have been provided, for ESs and EP, in [2]. There the model was for 40000 function evaluations, and the values obtained were $f^* \approx .4$ for and ES and $f^* \approx 200$ for EP. The niche search method reveals, therefore, a considerably improved performance. With 40000 evaluations, niche search obtains $f^* \approx 1.0 \cdot 10^{-13}$, $|\bar{x}_i^*| < 1. \cdot 10^{-07}, \forall i$.

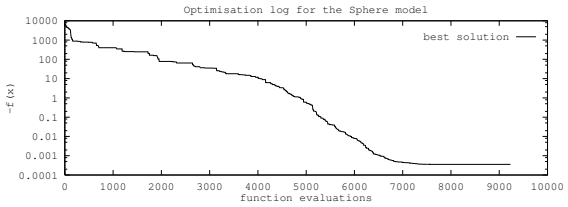

Fig. 5. Experimental run of niche search for the optimisation of the sphere model: best value obtained as a function of the number of objective evaluations.

3.2 Ackley function

Ackley test function is a strongly nonconcave and multimodal function, which has been proposed in [1].

$$\text{maximise } f(x) = 20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) + \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) - 20 - e$$
$$x_i \in [-30, 30], \quad n = 30$$

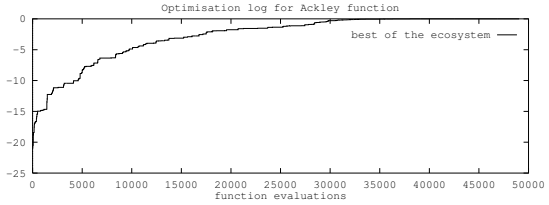

Fig. 6. Evolution log of an experimental run of niche search for the optimisation of Ackley's function: best value obtained as a function of the number of objective evaluations.

Parameters used by niche search for this test: 5600 generations, 3 niches with 3 individuals each. Average value of the solution found: $\bar{f}^* = -2.60 \cdot 10^{-9}$, $|\bar{x}_i^*| \leq 5.10 \cdot 10^{-10}$. Average number of function evaluations per run: 49539.

Experimental results for this test function have been provided, for ESs and EP, in [2]. They have run the model for 200000 function evaluations, and the values obtained are $f^* = 7.48 \cdot 10^{-08}$ for ES and $f^* = 1.39 \cdot 10^{-02}$ for EP. The niche search method has a considerably improved performance over both the systems, as approximately 25% of the evaluations were required to consistently obtain an equivalent solution. We also have run the model with close to 200000

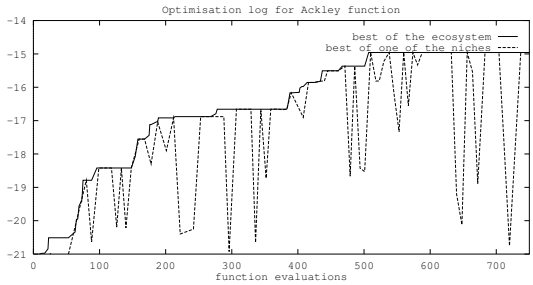

Fig. 7. Experimental run of niche search for the initial iterations of Ackley function's optimisation: best value obtained as a function of the number of objective evaluations, and best element of a particular niche. This shows how a particular niche can be far away form the best one during the search process.

function evaluations. The mean solution obtained was $\bar{f} = -3.5 \cdot 10^{-14}$, with $|x_i| < 1.4 \cdot 10^{-14}, \forall i$.

For a more complete set of benchmark tests, the reader is referred to [6].

4 Conclusion

Niche search is a genetic-based global optimisation solver that we have designed and implemented, trying to parallel closely the search operators on their biological equivalents. It is characterised by an evolutionary search on two layers: the individual layer and the niche layer. Neither of these searches is directed: both individuals and niches evolve based on the selection of the fittest.

The fact that different niches perform more localised searches in different regions of the search space strongly increases the probability of avoiding local optima. The possibility of using time adaption of the search parameters provides a way of improving the precision of the solution obtained, without drawbacks in terms of the global search. A penalisation method which allows for infeasible individuals to remain in the population was proposed. The aim of such a method is to increase the likeliness of finding the global optimum when it is located in narrow feasible regions; nevertheless, experimental support is not provided.

The numerical results obtained by niche search are quite promising, as our implementation has successfully handled all the tests carried out. The computational performance is considerably better than that of other algorithms of the same family analysed in the literature.

References

1. D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, Boston, 1987.

2. Thomas Bäck, Gúnter Rudolph, and Hans-Paul Schwefel. Evolutionary programming and evolution strategies: similarities and differences. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, 1993.
3. K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.
4. J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions od Systems, Man, and Cybernetics*, 16(1):122–128, 1986.
5. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, second, extended edition, 1994.
6. J. P. Pedroso. *Universal Service: Issues on Modelling and Regulation.* PhD thesis, Université Catholique de Louvain, 1996.
7. J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–59, San Mateo, CA, 1989.
8. R. Weinberg. *Computer simulation of a living cell.* PhD thesis, University of Michigan, 1970.

This article was processed using the LaTeX macro package with LLNCS style